网络流入门

2019年8月11日 18:39

一. 网络：指定了源点S和汇点T的带权有向图，边权c称作边容量

1. 流函数f(x,y)：定义在网络图点集上的实数函数，满足：
   i. 容量限制：f(x,y) <= c(x,y)
   ii. 斜对称：f(x,y) == - f(y,x)
   iii. 流量守恒：Σf(u,x) == Σf(x,v)，其中x是非S非T任意点，uv是x前驱后继
2. 关于流量的概念
   i. 流量：f(x,y)称为边(x,y)的流量
   ii. 剩余流量：c(x,y) - f(x,y)称为边(x,y)的剩余流量
   iii. 总流量：Σf(S,v)，其中S是源点，v是S后继
   iv. 根据流量守恒性质，易知非源点非汇点的点不会“存储流”，只是让流从源点经过它间接“流向”汇点而已
3. 其他
   i. 如果边(x,y)不在网络边集中，则默认容量c(x,y)=0
   ii. 根据斜对称性质，流量f(x,y)=|c|，iff 流量f(y,x)= -|c|，根据上一条性质所说的默认容量为0，一般也能保证反向边的负流量满足容量限制性质
   iii. 一般习惯对每条边用f/c的性质标注流量/容量
4. 网络流模型
   i. 最大流：使网络总流量最大的合法f函数值指派
   ii. 最小割：使S和T不连通，所需删去的边容量总和最少的边==最大流
   iii. 费用流：让每条边(x,y)的流量变化f时，需要花费f*w(x,y)的费用，求使流量最大的最小总费用或最大总费用
5. 增广路：从S到T的、每条边剩余容量都>0的路
   i. 残量网络：只包含增广路边的生成子图
   ii. 分层图：满足S到y的最短路恰比x多一条边的边(x,y)构成的生成子DAG
   iii. Edmonds-Karp算法：不断BFS残量网络找增广路，设增广路上最小剩余容量为minf，则网络最大流可增加minf。记得考虑反向边
   iv. Dinic算法：不断BFS求最短路，并构造分层图，再在分层图上DFS求增广路，并回溯更新剩余容量

二. 算法

1. Edmonds-Karp算法，最糟O（nm^2），适合1e3~1e4

```
const int MN = 1e4 + 5;

int n,m,tot; //点数、边数、边号
int s,t;     //源点，汇点
int fst[MN],val[MN<<5],to[MN<<5],nxt[MN<<5];
inline void add(int x,int y,int v){
     to[++tot] =y, val[tot] =v, nxt[tot] =fst[x], fst[x] =tot;
}      //用法是add(x,y,v),add(y,x,0);
```

```
queue<int>q; //bfs用的队列
int inq[MN],incf[MN],pre[MN];    //队内标记，各点为终点时的最小剩余容量，增广路上各点前驱边号
bool ekbfs(){//返回有没有搜到汇点出发到源点的增广路
    ms(inq,0);
    while(q.size())           q.pop();
    q.push(s),   inq[s] =1;
    incf[s] = 1<<29;
    while(q.size()){
        int x= q.front();  q.pop();
        for(int i= fst[x]; i; i= nxt[i]){
            if(!val[i])  continue;    //忽视无剩余容量的边
            int y= to[i];
            if(inq[y])   continue;    //只入队一次
            incf[y]= min(incf[x], val[i]);
            pre[y] =i;   //为了找最长路
            q.push(y),   inq[y] =1;
            if(y==t)     return 1;}}
    return 0;
}

ll maxflow;         //全局维护最大流总流量
inline void update(){     //更新增广路及其反向边的剩余容量
    int dx = incf[t];   //这次要修改的流量值
    for(int i, x =t; x!= s; x= to[i^1])    //用边号反向遍历增广路
        i = pre[x],  //找到通向该点的边号
        val[i] -= dx,
        val[i^1] += dx;     //反向边对称的
    maxflow += dx;
}

while(~scanf("%d%d",&n,&m)){
    ms(fst,0);
    s = 1, t = n, tot = 1,     maxflow = 0;//scanf("%d%d",&s,&t);
    int x,y,v;
    while(m--)
        scanf("%d%d%d",&x,&y,&v),
        add(x,y,v),
        add(y,x,0);
    while(ekbfs())            update();
    printf("%lld\n",maxflow);
}
```

2. Dinic算法，最糟O（n^2m），适用于1e4~1e5

```
const int MN = 1e5 + 5;

int n,m,tot; //点数、边数、边号
int s,t;     //源点，汇点
int fst[MN],cur[MN],val[MN<<5],to[MN<<5],nxt[MN<<5];
inline void add(int x,int y,int v){
    to[++tot] =y, val[tot] =v, nxt[tot] =fst[x], fst[x] =tot;
}     //用法是add(x,y,v),add(y,x,0);

queue<int>q; //bfs用的队列
int dst[MN]; //到源点最短路
bool dbfs(){ //求最短路，方便判断分层图中点的层级，返回能否届到汇点
```

```
    for__(x,1,n) cur[x] = fst[x];    //重置“当前弧”优化
    ms(dst,0);
    while(q.size())           q.pop();
    q.push(s),   dst[s] =1;
    while(q.size()){
        int x= q.front();  q.pop();
        for(int i= fst[x]; i; i= nxt[i]){
            if(!val[i])  continue;    //忽视无剩余容量的边
            int y= to[i];
            if(dst[y])   continue;    //bfs能保证一次求出最短路
            q.push(y),   dst[y]= dst[x] +1;
            if(y==t)     return 1;}}
    return 0;}

int dinic(int x, int Mflow){     //分层图上dfs找增广路，返回流量
    if(x==t)     return Mflow;
    int rt = Mflow;     //后半段增广路新增的流量，其初值是前半段容量
    for(int i= cur[x]; i && rt; i= nxt[i]){
        cur[x] =i;   //下一次使用当前x时，从未增广过的i开始遍历
        if(!val[i])  continue;    //忽视无剩余容量的边
        int y= to[i];
        if(dst[y] != dst[x]+1)    continue;    //忽视非残量网络边
        int dx= dinic(y, min(rt, val[i]));     //注意参数二取min
        if(!dx) dst[y]=0;  //剪枝，本次dfs不再考虑该y
        val[i] -= dx;val[i^1] += dx;           rt -= dx;}
    return Mflow - rt;  //本次dfs增加的后半段的容量
}
ll maxflow;         //全局维护最大流总流量
inline void update(){     //更新最近一次dinic返回的新增流量
    for(int dx; dx= dinic(s, 1<<30); )
        maxflow += dx;
}

while(~scanf("%d%d",&n,&m)){
    ms(fst,0);
    s = 1, t = n, tot = 1,     maxflow = 0;//scanf("%d%d",&s,&t);
    int x,y,v;
    while(m--)
        scanf("%d%d%d",&x,&y,&v),
        add(x,y,v),
        add(y,x,0);
    while(dbfs())update();    //残量网络能届到汇点就更新
    printf("%lld\n",maxflow);
}
```